GPU TECHNOLOGY CONFERENCE

TENSOR CORE PROGRAMMABILITY AND PROFILING FOR AI AND HPC APPLICATIONS

Griffin Lacey

Max Katz

PRESENTED BY NVIDIA

TOO LONG; DIDN'T LISTEN

- Tensor Cores enable fast **mixed precision matrix multiplications**
- Growing number of AI/HPC examples **accelerated up to 25x**
- Mature software support with **high-level APIs and Nsight developer tools**
- All you need is **Volta / Turing GPU**

OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores

OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores

FIRST, WHAT IS PRECISION?

- Precision is a measure of numerical detail
- Floating Point (FP) is a representation of real numbers supporting the tradeoff of:
  - Precision (significand)
  - Range (exponent)
- Lower precision numbers have computational performance advantages

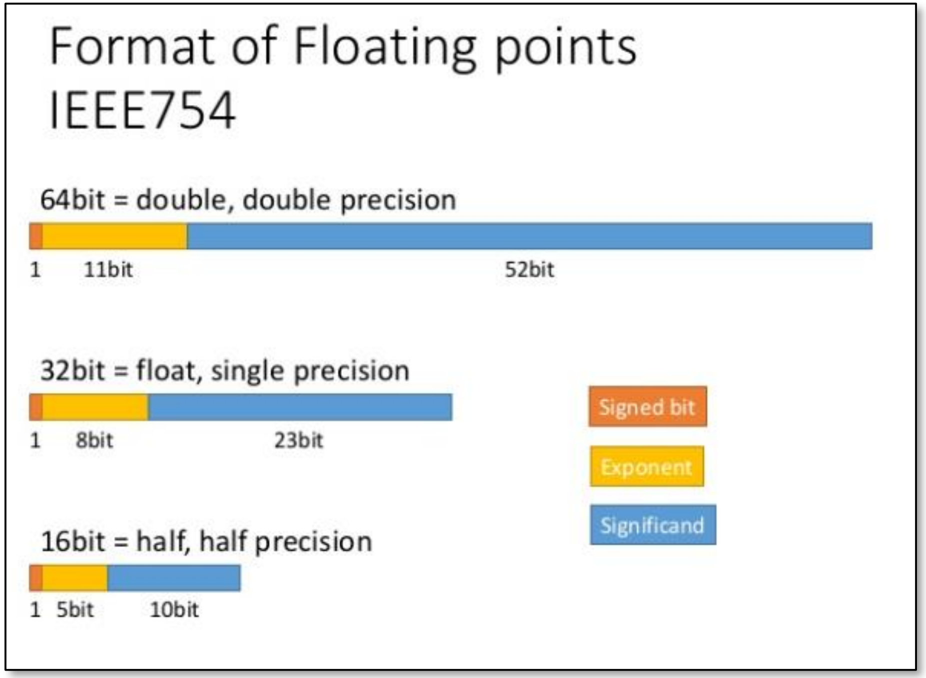
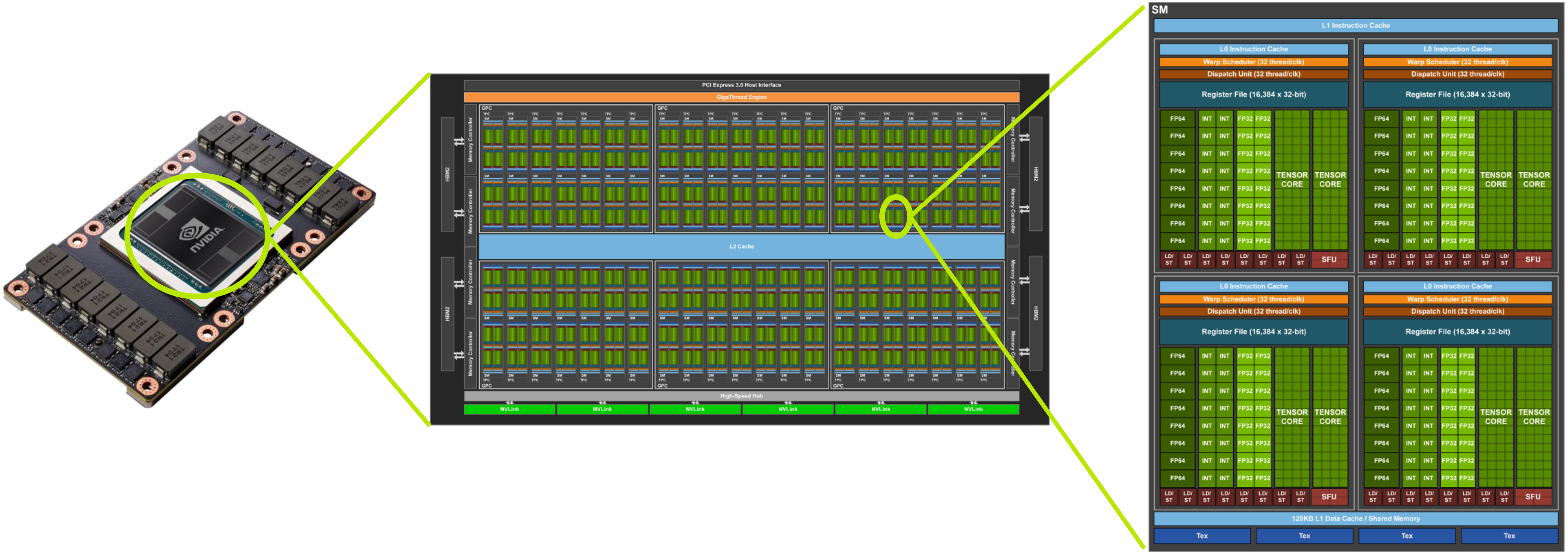


Figure: https://devblogs.nvidia.com/tensor-cores-mixed-precision-scientific-computing/

WHAT ARE TENSOR / CUDA CORES?



Figures: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

VOLTA GV100 SM

	GV100
FP32 units	64
FP64 units	32
INT32 units	64
Tensor Cores	8
Register File	256 KB
Unified L1/Shared memory	128 KB
Active Threads	2048

CUDA CORES

VOLTA TENSOR CORE

Half/Mixed Precision 4x4 Matrix Multiply-Accumulate

$$D = AB + C$$

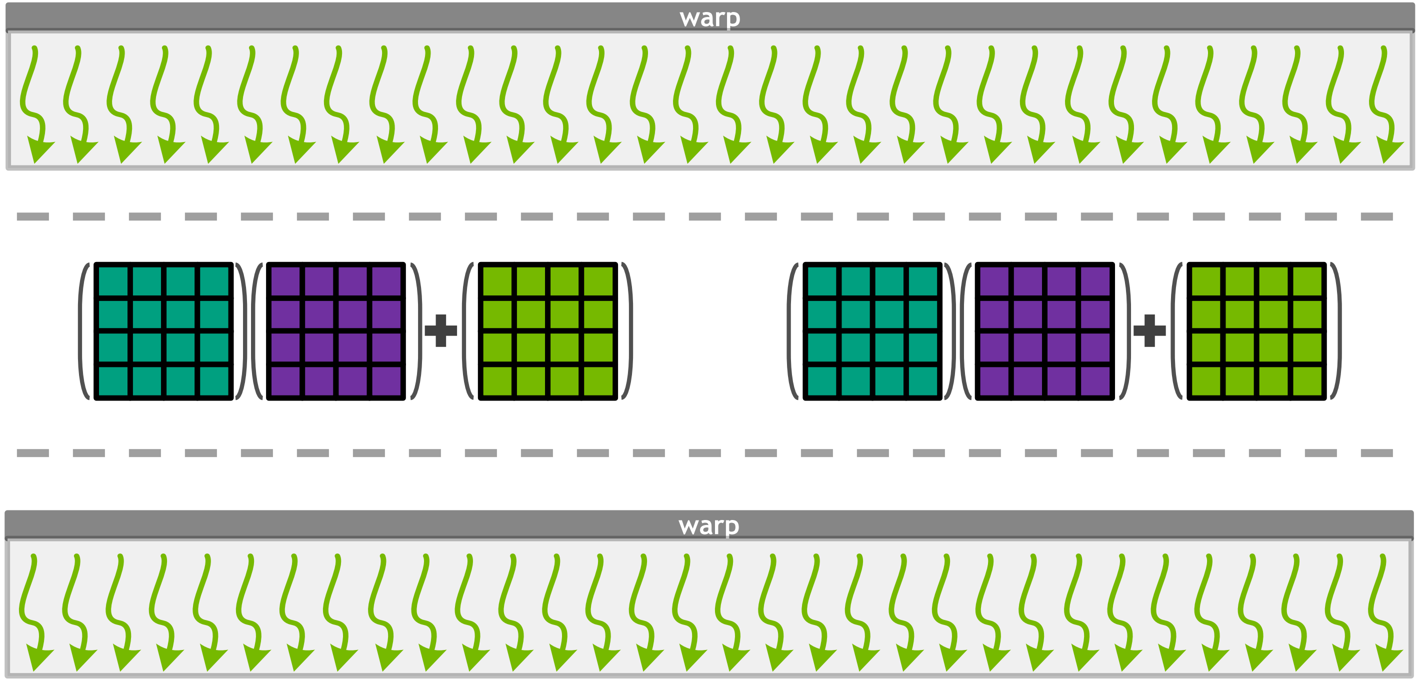
$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 | FP16 | FP16 | FP16 or FP32

Turing Tensor Cores: support for int8, int4

VOLTA TENSOR CORE

Full Warp 16x16 Matrix Math



Warp-synchronizing operation for cooperative matrix math

Aggregate Matrix Multiply and Accumulate for **16x16** matrices

Result distributed across warp

OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores

TENSOR CORES FOR AI

- Simple trick for 2x to 5x faster deep learning training
  - Accomplished in few lines of code
  - Models can use same hyperparameters
  - Models converge to same accuracy
- Half the memory traffic and storage enabling larger batch sizes
- AI community is trending towards low precision as common practice

HOW TO USE TENSOR CORES

CUDA

- Exposed as instructions in CUDA under WMMA API (**W**arp **M**atrix **M**ultiply **A**ccumulate)

Low Level Libraries

- Used by cuDNN, cuBLAS, CUTLASS to accelerate matrix multiplications and convolution

Deep Learning Frameworks

- Tensor Core kernels used implicitly on FP16 ops from DL frameworks PyTorch / TensorFlow / etc…

High Level APIs

- High-level tools (e.g. PyTorch Apex) convert everything automatically and safely

MIXED PRECISION TRAINING

1. Model conversion
2. Master weight copy
3. Loss scaling

1. MODEL CONVERSION

- Make simple type updates to each layer:
  - Use FP16 values for the weights and inputs

```
# PyTorch
layer = torch.nn.Linear(in_dim, out_dim).half()

# TensorFlow
layer = tf.layers.dense(tf.cast(inputs, tf.float16), out_dim)
```

2. MASTER WEIGHTS

- FP16 alone is sufficient for some networks but not others; keep FP32 copy of weights

```
param = torch.cuda.FloatTensor([1.0])
print(param + 0.0001)
```

→ 1.0001

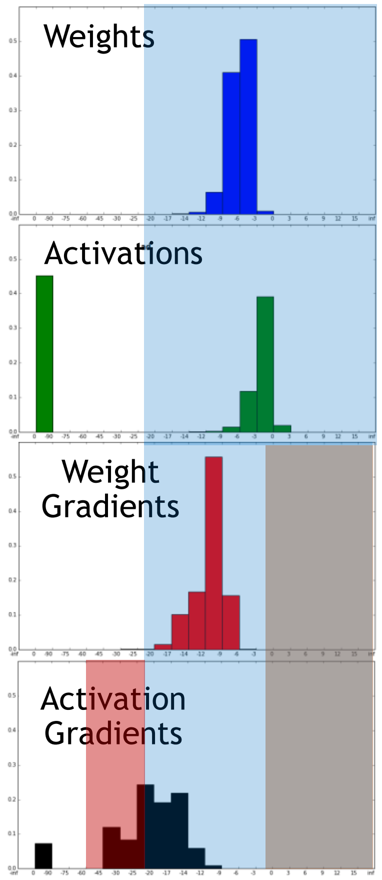
```
param = torch.cuda.HalfTensor([1.0])
print(param + 0.0001)
```

→ 1

When *update/param* < 2^{-11}, updates have no effect.

3. LOSS SCALING

- Range representable in FP16: ~40 powers of 2
- Gradients are small:
  - Some lost to zero
  - While ~15 powers of 2 remain unused
- Loss scaling:
  - Multiply loss by a constant S
  - All gradients scaled up by S (chain rule)
  - Unscale weight gradient (in FP32) before weight update



MIXED PRECISION TRAINING

1. Model conversion
2. Master weight copy
3. Loss scaling

Automated Mixed Precision (AMP)
(e.g. PyTorch Apex)

PYTORCH APEX AMP 1.0

```
N, D_in, D_out = 64, 1024, 512
x = Variable(torch.randn(N, D_in )).cuda()
y = Variable(torch.randn(N, D_out)).cuda()

model = torch.nn.Linear(D_in, D_out).cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)


for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    optimizer.zero_grad()

    loss.backward()
    optimizer.step()
```

PYTORCH APEX AMP 1.0

```
N, D_in, D_out = 64, 1024, 512
x = Variable(torch.randn(N, D_in )).cuda()
y = Variable(torch.randn(N, D_out)).cuda()

model = torch.nn.Linear(D_in, D_out).cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")

for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    optimizer.zero_grad()
    with amp.scale_loss(loss, optimizer) as scaled_loss:
        scaled_loss.backward()
    optimizer.step()
```

MIXED PRECISION SPEEDUPS

Not Limited to Image Classification

Model	FP32 -> FP16 Speedup	Comments
GNMT (Translation)	2.3x	Iso-batch size
FairSeq Transformer (Translation)	2.9x 4.9x	Iso-batch size 2x lr + larger batch
ConvSeq2Seq (Translation)	2.5x	2x batch size
Deep Speech 2 (Speech recognition)	4.5x	Larger batch
wav2letter (Speech recognition)	3.0x	2x batch size
Nvidia Sentiment (Language modeling)	4.0x	Larger batch

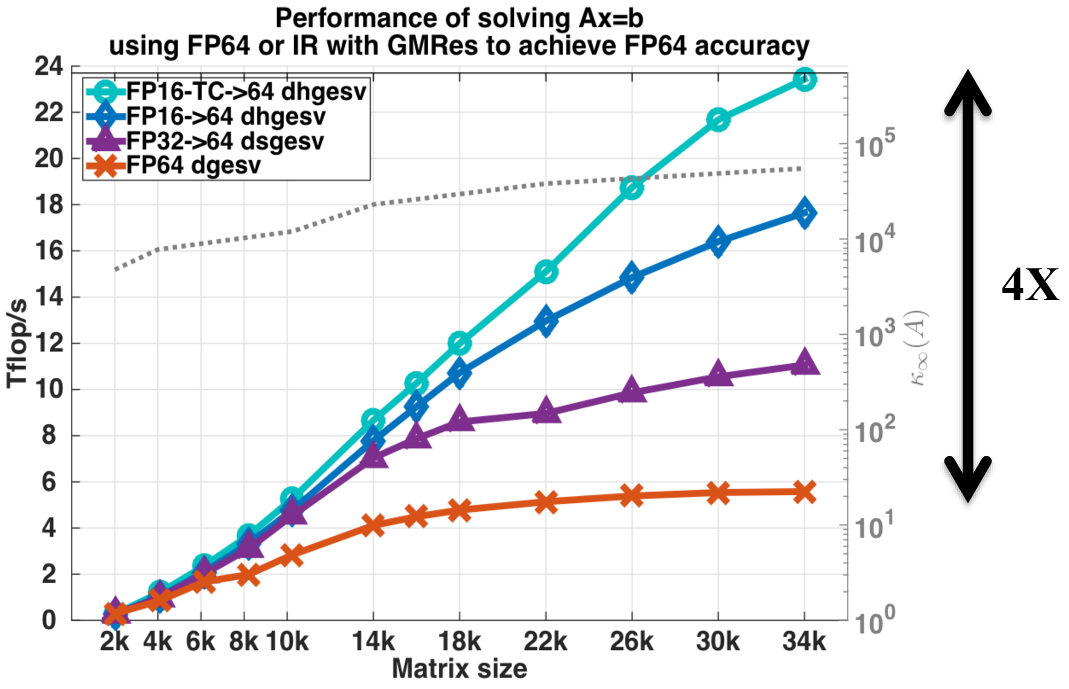
OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores

TENSOR CORES FOR HPC

- Mixed precision algorithms are increasingly popular
  - It is common to combine double + single precision, or floating point + integer
- Similar to AI:
  - Use low precision to reduce memory traffic and storage
  - Use Tensor Core instructions for large speedups

LINEAR ALGEBRA



- Researchers from ICL/UTK
- Accelerated FP64 LU factorization 4x using Tensor Cores in MAGMA
- Compute initial solution in FP16, then iteratively refine solution
- Achieved FP64 TFLOPS: 5.8
- Achieved FP16->FP64 TFLOPS: 24

Data courtesy of: Azzam Haidar, Stan. Tomov & Jack Dongarra, Innovative Computing Laboratory, University of Tennessee
"*Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers*", A. Haidar, S. Tomov, J. Dongarra, N. Higham SC'18
GTC 2018 Poster P8237: *Harnessing GPU's Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solves*

INNOVATIVE COMPUTING LABORATORY MAGMA
THE UNIVERSITY OF TENNESSEE

EARTHQUAKE SIMULATION

- Researchers from University of Tokyo, Oak Ridge National Laboratory (ORNL), and the Swiss National Supercomputing Centre
- Solver called MOTHRA achieved 25x compared to standard solver
- Used AI to identify where to apply low or high precision in solver
- Used a combination FP64, FP32, FP21 and FP16 to further reduce computational and communication costs

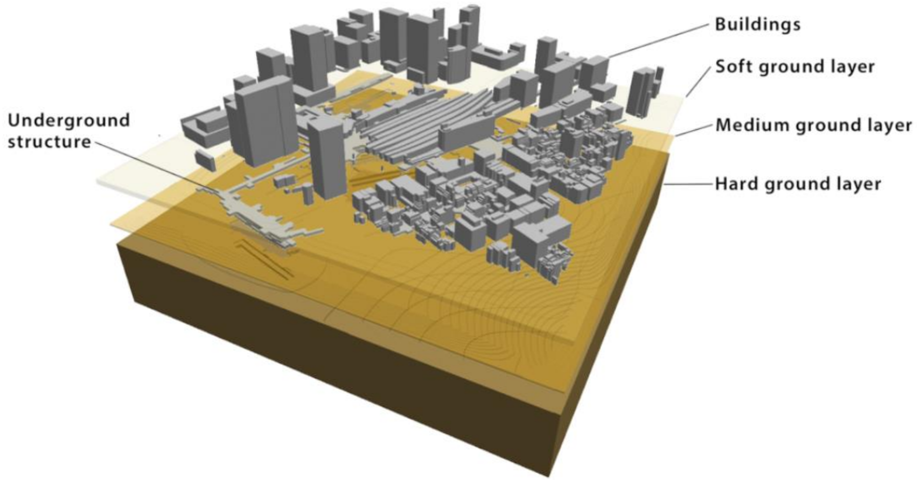


Figure 3. Earthquake simulation model used by the Supercomputing 2018 (SC'18) Gordon Bell finalist paper for Tokyo including layers of soil, underground and above ground structures, and the coupling between them.

Paper: "A fast scalable implicit solver for nonlinear time-evolution earthquake city problem on low-ordered unstructured finite elements with artificial intelligence and transprecision computing", Ichimura et. al, SC 18.

OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores

NSIGHT DEVELOPER TOOLS

NSIGHT PRODUCT FAMILY

Standalone Performance Tools

Nsight Systems - System-wide application algorithm tuning

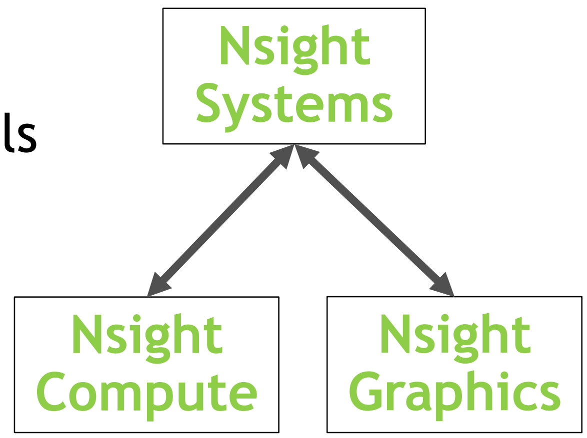
Nsight Compute – Debug CUDA API and optimize CUDA kernels

Nsight Graphics - Debug/optimize specific graphics apps

IDE Plugins

Nsight Eclipse Edition/Visual Studio – editor, debugger, some perf analysis

Workflow



NSIGHT SYSTEMS

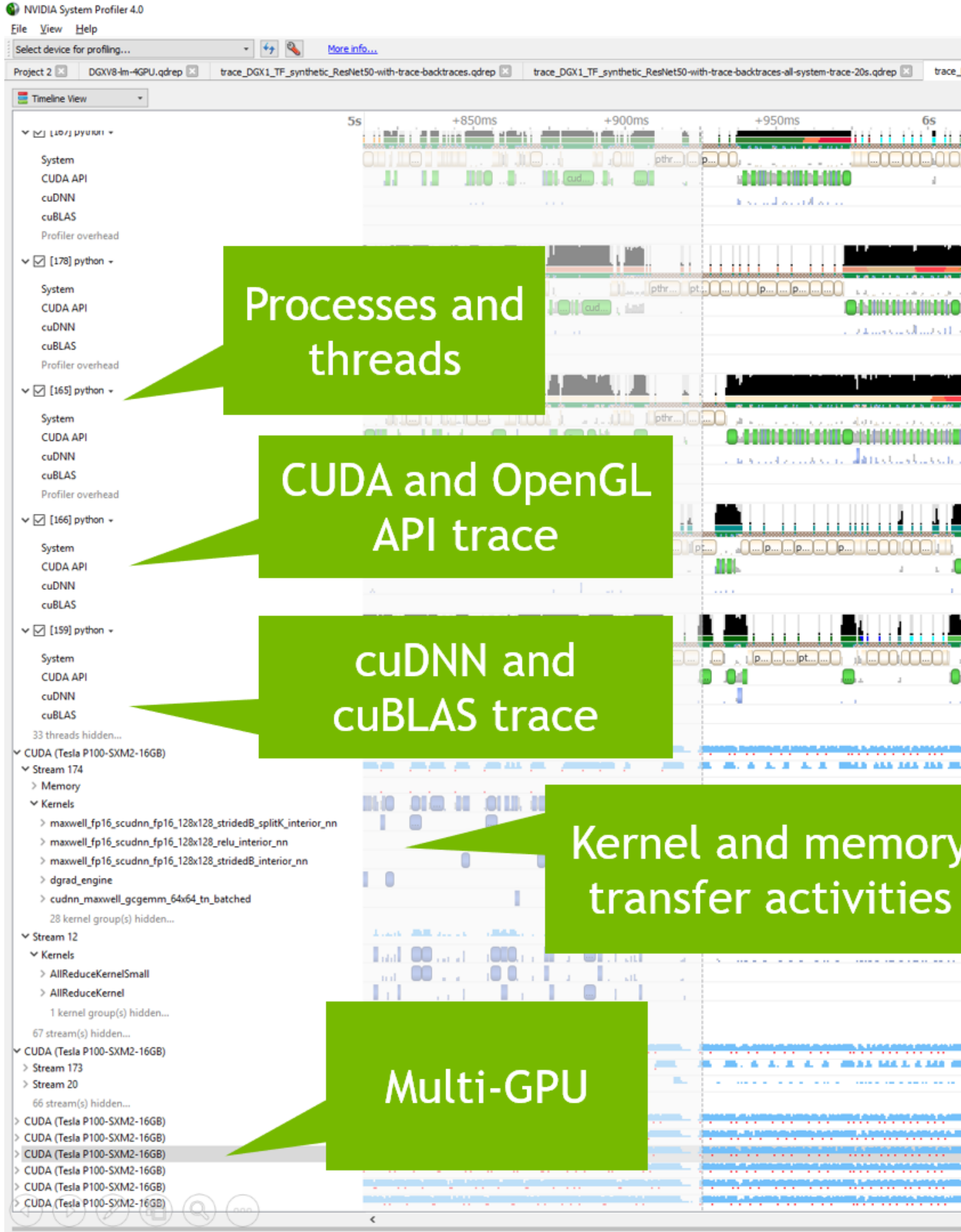
Next-Gen System Profiling Tool

System-wide application algorithm tuning
- Multi-process tree support

Locate optimization opportunities
- Visualize millions of events on a fast GUI timeline
- Or gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs
- CPU algorithms, utilization, and thread state
- GPU streams, kernels, memory transfers, etc

Multi-platform: Linux & Windows, x86-64 & Tegra, MacOSX (host only)



NSIGHT COMPUTE

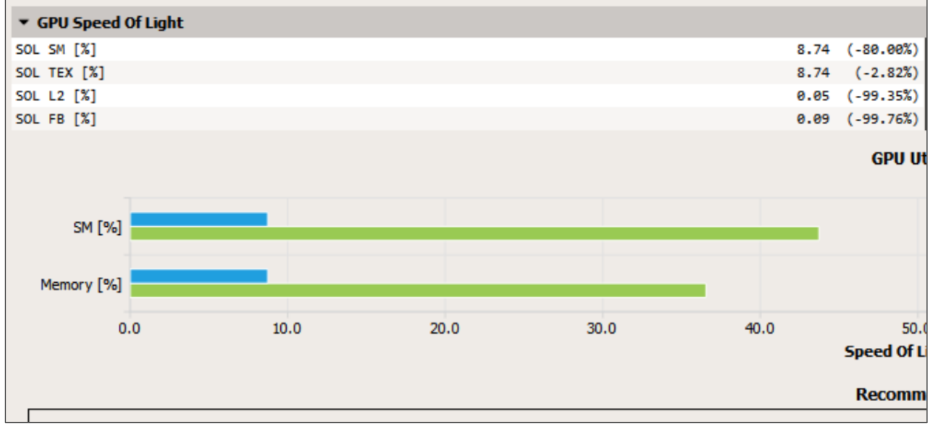
Next-Gen Kernel Profiling Tool

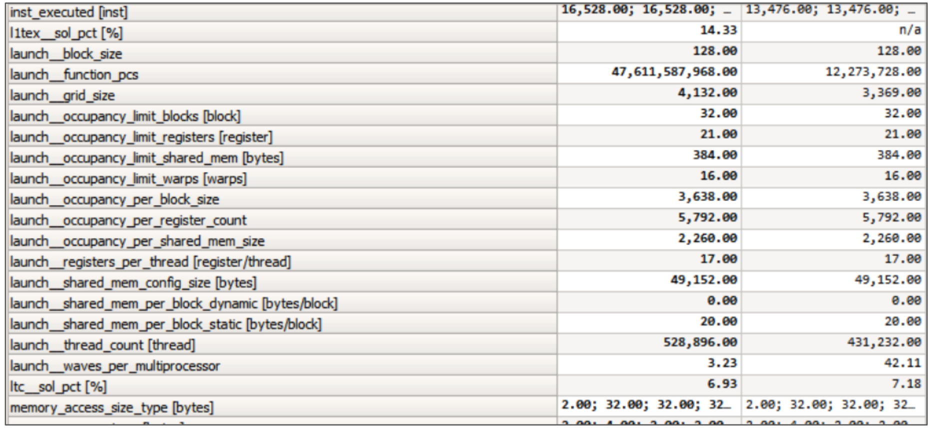
Key Features:

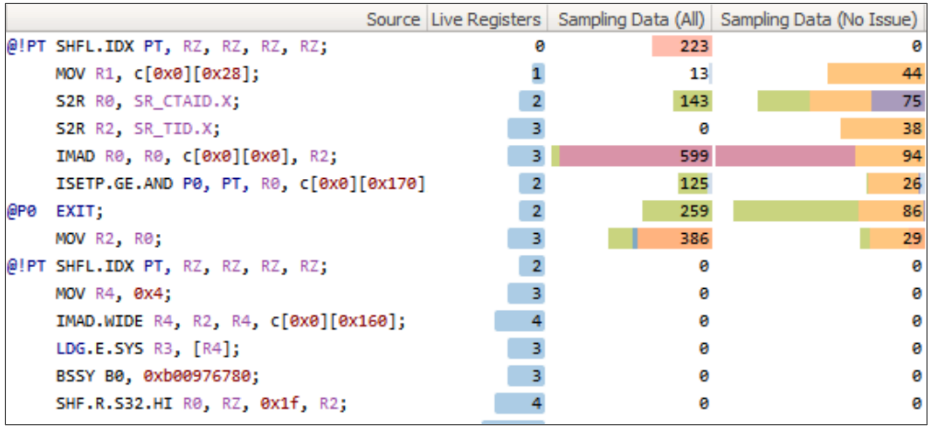
- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Improved Workflow (diffing results)
- Fully Customizable (programmable UI/Rules)
- Command Line, Standalone, IDE Integration

OS: Linux, Windows, ARM, MacOSX (host only)

GPUs: Pascal (GP10x), Volta, Turing







USING NSIGHT SYSTEMS

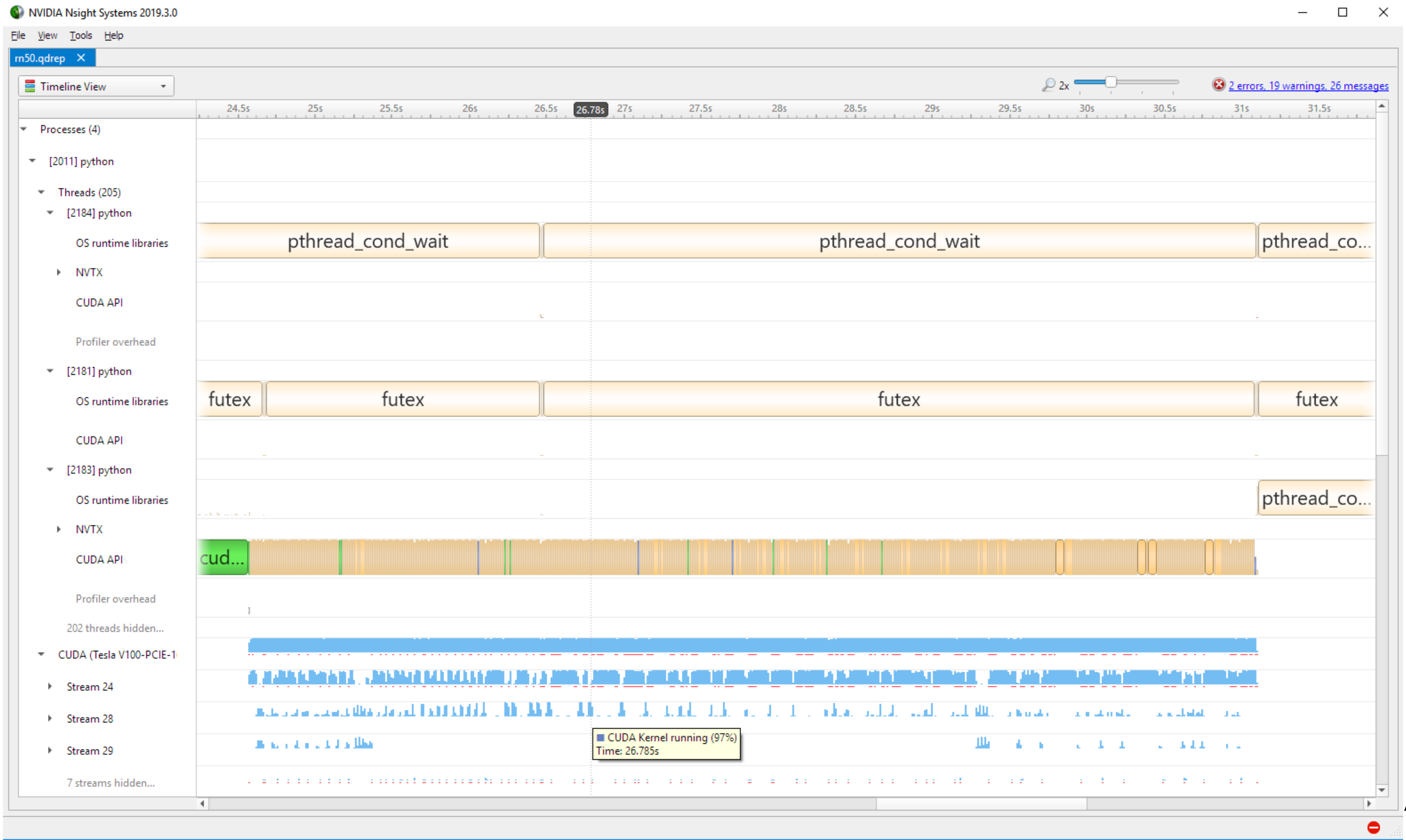
COLLECT A PROFILE WITH NSIGHT SYSTEMS

```
$ nsys profile /usr/bin/python train.py
```

Generated file: `report.qdrep`

Import for viewing into the Nsight Systems UI

The Nsight Systems UI can also be used for interactive system profiling


NVIDIA Nsight Systems 2019.3.0
File
View
Tools
Help
rn50.qdrep
Timeline View
2x
2 errors, 19 warnings, 26 messages
24.5s
25s
25.5s
26s
26.5s
26.78s
27s
27.5s
28s
28.5s
29s
29.5s
30s
30.5s
31s
31.5s
Processes (4)
[2011] python
Threads (205)
[2184] python
OS runtime libraries
NVTX
CUDA API
Profiler overhead
[2181] python
OS runtime libraries
CUDA API
[2183] python
OS runtime libraries
NVTX
CUDA API
Profiler overhead
202 threads hidden...
CUDA (Tesla V100-PCIE-1
Stream 24
Stream 28
Stream 29
7 streams hidden...
pthread_cond_wait
pthread_cond_wait
pthread_co...
futex
futex
futex
futex
pthread_co...
cud...
CUDA Kernel running (97%)
Time: 26.785s
A.

LOCATING TENSOR CORE KERNELS

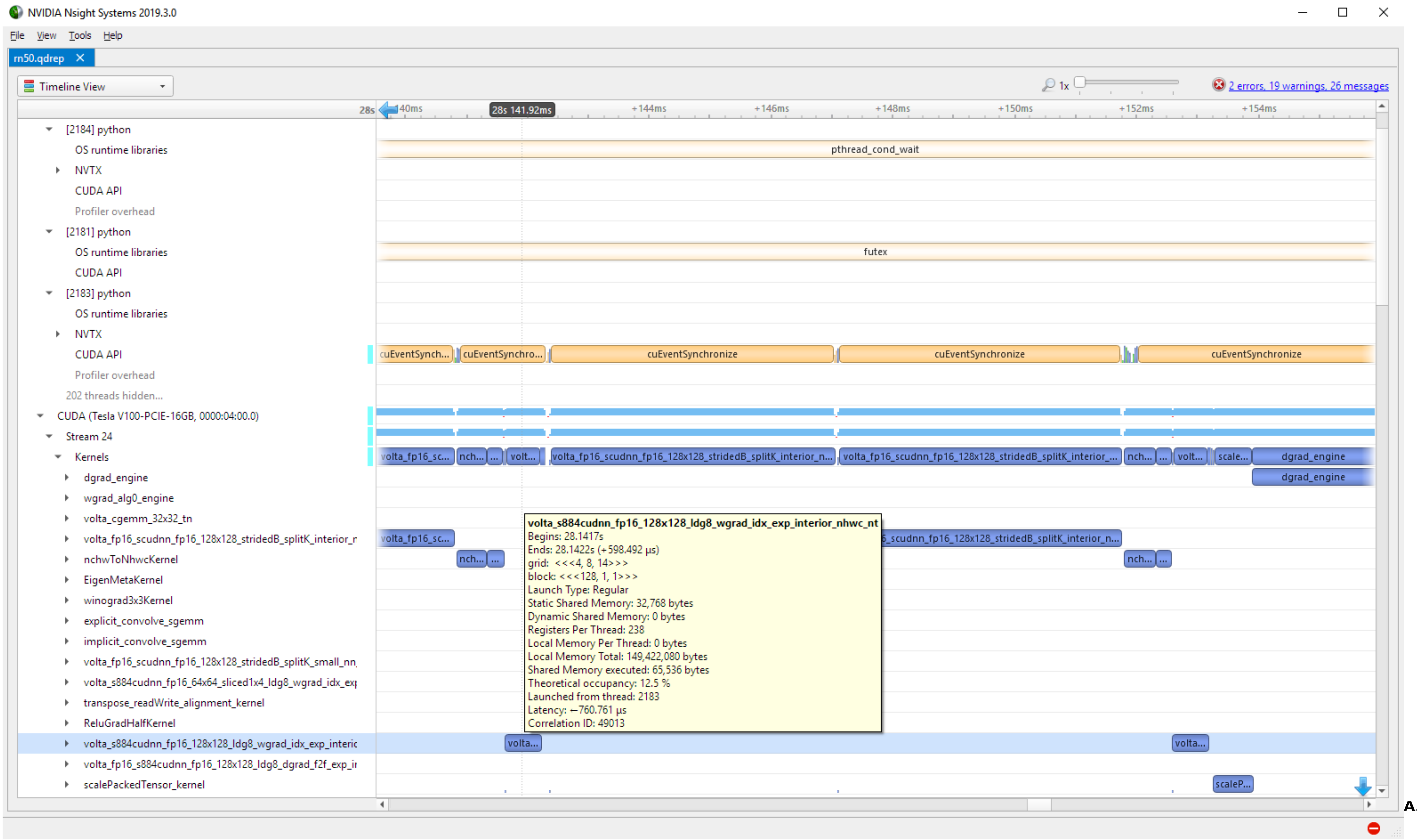
On Volta V100, CUDA kernels using tensor cores contain the string “s884”

Examples:

volta_fp16_**s884**gemm_fp16_128x64_ldg8_f2f_nn

volta_fp16_**s884**cudnn_fp16_256x64_ldg8_relu_f2f_exp_interior_nhwc2nchw_tn_v1

These are kernels with HMMA (half-precision matrix multiply and accumulate) machine instructions


NVIDIA Nsight Systems 2019.3.0
File View Tools Help
rn50.qdrep
Timeline View
1x
2 errors, 19 warnings, 26 messages
28s
40ms
28s 141.92ms
+144ms
+146ms
+148ms
+150ms
+152ms
+154ms
[2184] python
OS runtime libraries
NVTX
CUDA API
Profiler overhead
[2181] python
OS runtime libraries
CUDA API
[2183] python
OS runtime libraries
NVTX
CUDA API
Profiler overhead
202 threads hidden...
CUDA (Tesla V100-PCIE-16GB, 0000:04:00.0)
Stream 24
Kernels
dgrad_engine
wgrad_alg0_engine
volta_cgemm_32x32_tn
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_r
nchwToNhwcKernel
EigenMetaKernel
winograd3x3Kernel
explicit_convolve_sgemm
implicit_convolve_sgemm
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_small_nn
volta_s884cudnn_fp16_64x64_sliced1x4_ldg8_wgrad_idx_exp
transpose_readWrite_alignment_kernel
ReluGradHalfKernel
volta_s884cudnn_fp16_128x128_ldg8_wgrad_idx_exp_interic
volta_fp16_s884cudnn_fp16_128x128_ldg8_dgrad_f2f_exp_ir
scalePackedTensor_kernel
pthread_cond_wait
futex
cuEventSynch...
cuEventSynchro...
cuEventSynchronize
cuEventSynchronize
cuEventSynchronize
volta_fp16_sc...
nch...
volt...
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_n...
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_...
nch...
volt...
scale...
dgrad_engine
dgrad_engine
volta_fp16_sc...
_scudnn_fp16_128x128_stridedB_splitK_interior_n...
nch...
nch...
volta...
volta...
scaleP...
volta_s884cudnn_fp16_128x128_ldg8_wgrad_idx_exp_interior_nhwc_nt
Begins: 28.1417s
Ends: 28.1422s (+598.492 µs)
grid: <<<4, 8, 14>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 32,768 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 238
Local Memory Per Thread: 0 bytes
Local Memory Total: 149,422,080 bytes
Shared Memory executed: 65,536 bytes
Theoretical occupancy: 12.5 %
Launched from thread: 2183
Latency: ←760.761 µs
Correlation ID: 49013

A.

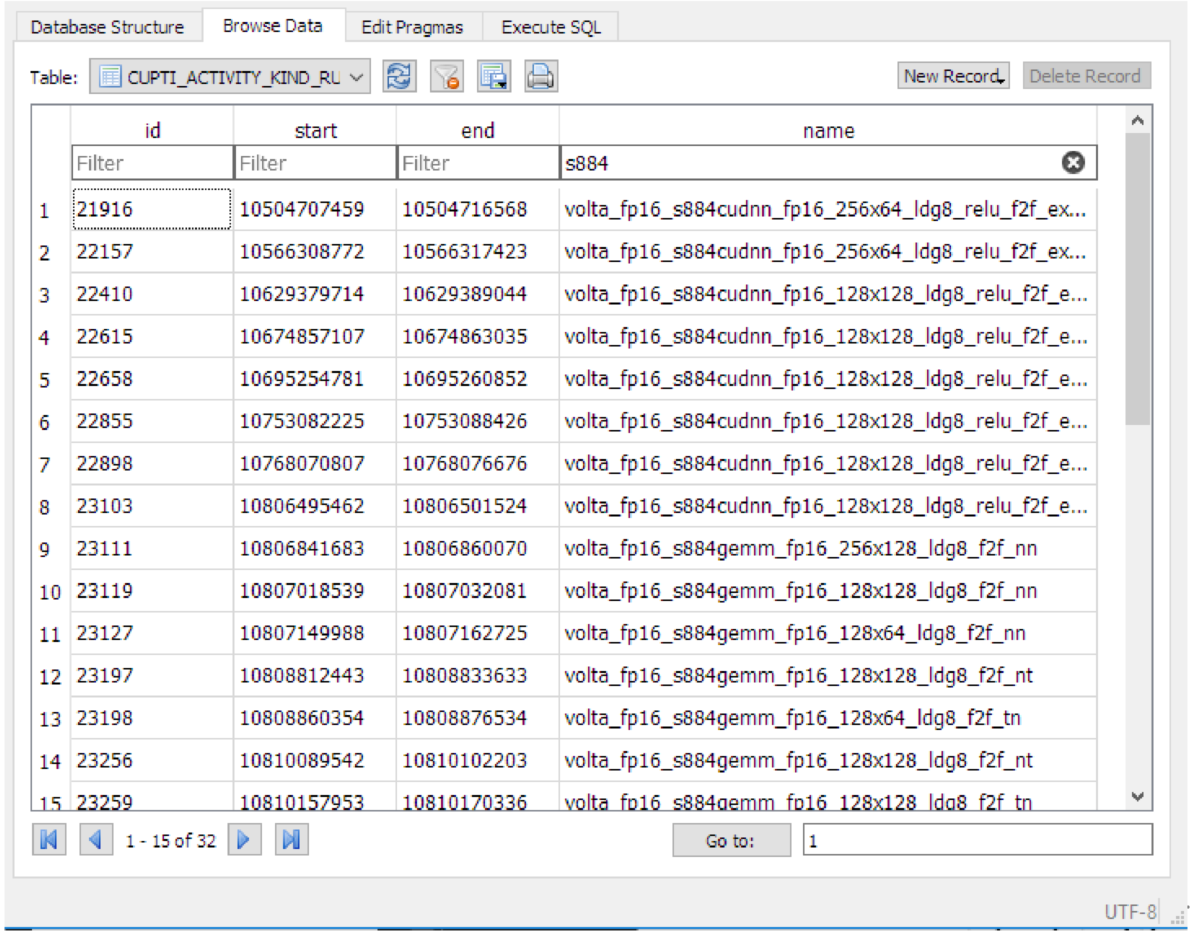
COMING SOON: SQLITE DATABASE EXPORT

USE NSYS-EXPORTER TO CREATE SQLITE DB

`nsys-exporter -s report.qdrep`

Generated DB: `report.sqlite`

Interact with this like any SQLite database



	id	start	end	name
	Filter	Filter	Filter	s884
1	21916	10504707459	10504716568	volta_fp16_s884cudnn_fp16_256x64_ldg8_relu_f2f_ex...
2	22157	10566308772	10566317423	volta_fp16_s884cudnn_fp16_256x64_ldg8_relu_f2f_ex...
3	22410	10629379714	10629389044	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_e...
4	22615	10674857107	10674863035	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_e...
5	22658	10695254781	10695260852	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_e...
6	22855	10753082225	10753088426	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_e...
7	22898	10768070807	10768076676	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_e...
8	23103	10806495462	10806501524	volta_fp16_s884cudnn_fp16_128x128_ldg8_relu_f2f_e...
9	23111	10806841683	10806860070	volta_fp16_s884gemm_fp16_256x128_ldg8_f2f_nn
10	23119	10807018539	10807032081	volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nn
11	23127	10807149988	10807162725	volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
12	23197	10808812443	10808833633	volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nt
13	23198	10808860354	10808876534	volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_tn
14	23256	10810089542	10810102203	volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_nt
15	23259	10810157953	10810170336	volta_fp16_s884gemm_fp16_128x128_ldg8_f2f_tn

ASSOCIATE KERNEL NAMES WITH EVENTS

```
ALTER TABLE CUPTI_ACTIVITY_KIND_RUNTIME ADD COLUMN name TEXT;

UPDATE CUPTI_ACTIVITY_KIND_RUNTIME SET name =

    (SELECT value FROM StringIds

    JOIN CUPTI_ACTIVITY_KIND_KERNEL AS cuda_gpu

        ON cuda_gpu.demangledName = StringIds.id

        AND CUPTI_ACTIVITY_KIND_RUNTIME.correlationId =
cuda_gpu.correlationId);
```

LOCATE KERNELS USING TENSOR CORES

```
SELECT
                *
FROM
                CUPTI_ACTIVITY_KIND_RUNTIME as cupti
WHERE
                cupti.name LIKE '%s884%'
```

USING NSIGHT COMPUTE

KERNEL PROFILES WITH NSIGHT COMPUTE

```
$ nv-nsight-cu-cli /usr/bin/python train.py
```

This is expensive for a typical DL training session because it will collect metrics for every kernel; consider profiling fewer kernels.

For example, to profile `*s884*` kernels on all streams, but only on the fifth invocation:

```
$ nv-nsight-cu-cli --kernel-id ::s884:5 /usr/bin/python train.py
```

The Nsight Systems UI can also be used for interactive kernel profiling

INTERLUDE: FINDING TENSOR CORE METRICS

Isolating which GPU metrics measure tensor cores:

```
$ nv-nsight-cu-cli --devices 0 --query-metrics | grep -i tensor
...
smsp__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active
sm__pipe_tensor_op_hmma_cycles_active.avg
sm__inst_executed_pipe_tensor.avg.per_second
...
```

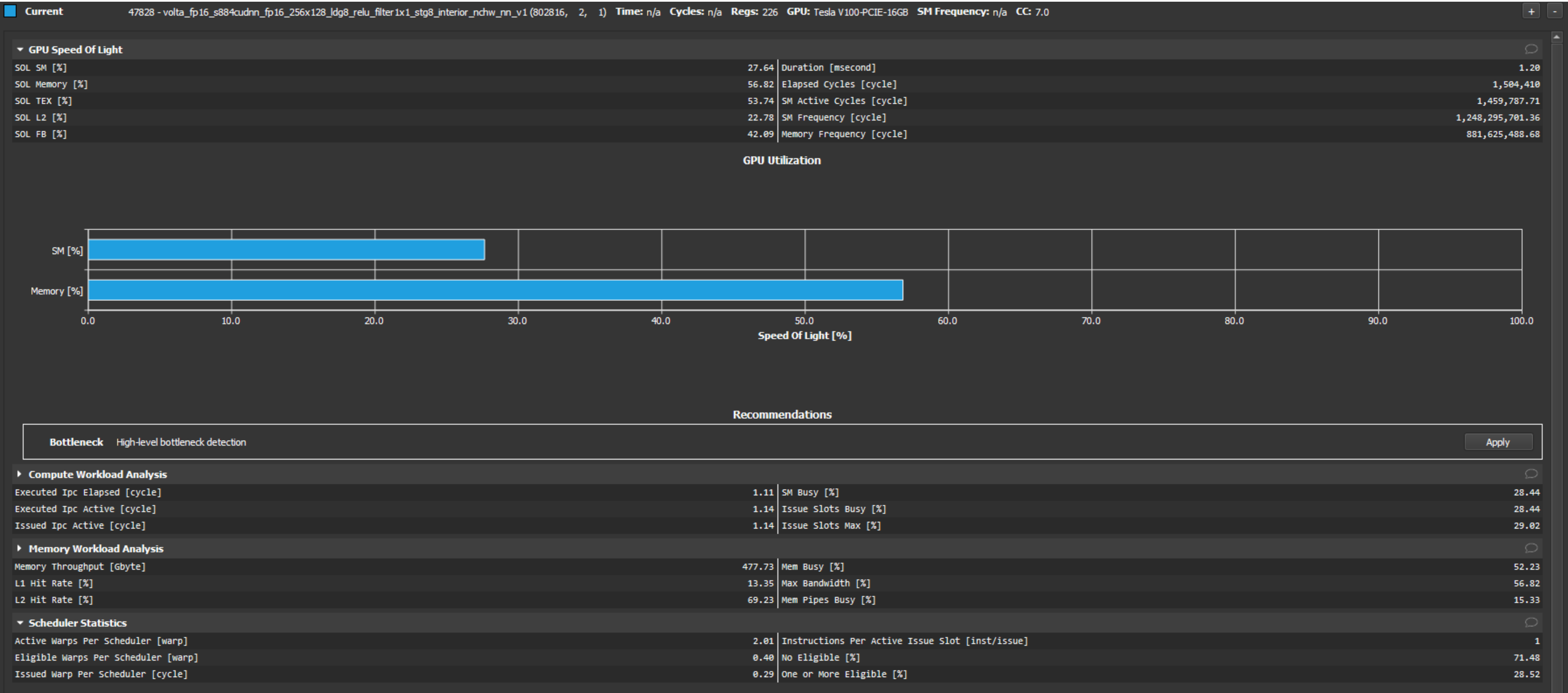
TENSOR CORE PERFORMANCE

Now add in what we know about which metrics to looks for:

```
$ nv-nsight-cu-cli --kernel-id ::s884:5 --metrics
smsp__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active
/usr/bin/python train.py
```

```
volta_s884cudnn_fp16_128x128_ldg8_wgrad_idx_exp_interior_nhwc_nt, 2019-Mar-17 02:54:29, Context 1, Stream 23

    Section: Command line profiler metrics

    ----------------------------------------------------------------------- --------------- ------------------------------

    smsp__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active                    %                          79.03

    ----------------------------------------------------------------------- --------------- ------------------------------
```

NSIGHT COMPUTE UI



SUMMARY

NVIDIA TOOLS FOR TENSOR CORE PROFILING

System, application, and kernel level profiling solutions

- **Nsight Systems**: high-level application view; locate kernels that used tensor cores
- **Nsight Compute**: drill down into specific kernels for detailed performance analysis
- Starting with version 19.03, NVIDIA GPU Cloud (NGC) optimized deep learning containers package Nsight Systems and Nsight Compute. Download and try it now!
- Coming soon: DLProf tool for connecting tensor core usage with popular DL frameworks. More information: S9339 – Profiling Deep Learning Networks @ GTC 2019

DEVELOPER TOOLS AT GTC19

Talks:
S9751: Accelerate Your CUDA Development with Latest Debugging and Code Analysis Developer Tools, Tue @9am
S9866 - Optimizing Facebook AI Workloads for NVIDIA GPUs, Tue @9am
S9345: CUDA Kernel Profiling using NVIDIA Nsight Compute, Tue @1pm
S9661: Nsight Graphics - DXR/Vulkan Profiling/Vulkan Raytracing, Wed @10am
S9503: Using Nsight Tools to Optimize the NAMD Molecular Dynamics Simulation Program, Wed @1pm

Hands-on labs:
L9102: Jetson Developer Tools Training Lab, Mon @9am, 11:30am
L9124: Debugging and optimizing CUDA applications with Nsight products on Linux training lab, Tue @8am, 10am

Connect with the Experts (where DevTools will be available):
CE9123: CUDA & Graphics Developer Tools, Tue @2pm, Wed @3pm
CE9137: Jetson Embedded Platform, Tue @12pm, 5pm, Wed @1pm, 4pm, Thu @12pm

Podium: Demos of DevTools products on Linux, DRIVE AGX & Jetson AGX at the showfloor
Tue @12pm – 7pm
Wed @12pm – 7pm
Thu @11am – 2pm

PRESENTED BY NVIDIA

GPU TECHNOLOGY CONFERENCE

SAN JOSE | MAY 17-21, 2019

https://www.nvidia.com/en-us/gtc/

#GTC19